SWFModeller

# Summary

SWFModeller provides functionality that allows SWF files to be created, loaded, modified and saved. Using SWFModeller classes you can create and piece together a new SWF file entirely in memory and save it to a new SWF file. Alternatively you can load an existing SWF file, modify it and re-save it as a new SWF.

A handy reference to have might be the SWF file format specification: <http://www.adobe.com/devnet/swf.html>

Note that SWFModeller was created using the version 9 spec.

# Class Overview

SWFModeller functionality is roughly split into IO, Modelling and ABC bytecode areas.



Figure Simplified class overview

Briefly summarised, the SWFModeller represents a SWF as a single object extending Timeline. A timeline is composed of frames which are each simply lists of instructions (IDisplayListItem objects) that either put things onto or remove things from the stage.

The items placed onto the stage are known as characters which can either be shapes or Sprite objects (a.k.a MovieClips).

## Quirks

Throughout the code there is a trade-off apparent between rapid development and realisation of an ultimate goal. The most common trade-off is between representing the SWF internally in a malleable way and representing it in a way that allows it simply to be recreated as a file.

A good example is frames and layers. For the sake of rapid development they are represented in a way which allows them to be written back out to file again without any attempt at understanding the data in a useful way. Ultimately more work will ideally be done to make SWFModeller a more complete library so that frames and layers can be manipulated in meaningful ways but you should be away that for now things are a work in progress. Things that seem like strange decisions may well be placeholders until something better comes along.

# Characters

Characters are things that appear on stage. These are things like sprites and vector shapes.



Characters are referenced on the stage through their ICharacter interface. At the moment, any concrete implementation of a shape will be either an UnparsedShape object, or an UnparsedMorphShape object. These are thin wrappers around the raw binary shape data taken from the source SWF file. When the shape is referenced on the stage it is represented by a proxy object. This means that in the future if we find the need to parse the shape data then it can be done on-demand. At this point the object can be swapped out for a parsed shape object and we won’t have to scour the stage for references to our shape; we just update the proxy and all references will point to the new object.

At the moment though, we don’t have any need to parse shapes, so this is a ‘just in case’ complication, and will probably only slow you down and bemuse you. Sorry.

# SWF Parsing

TODO

# SWF Writing

TODO

# Display list

TODO

# ABC Bytecode

TODO

## Quirks

TODO: On-demand parsing only gets in the way.

# Exception handling

TODO

# Testing

TODO

## Quirks

TODO: Output order is different to input order: The ‘Export on frame 1’ problem.